Subject: Modifying Arrays and Structures in HASH's (hint: you can't)
Posted by JDS on Thu, 22 Mar 2012 21:58:19 GMT
View Forum Message <> Reply to Message

HASH's are nice in that they can contain any variable type, and make iterating over deeply nested data structure more humane. On the other hand, their limitations as separate standalone objects, and not as a deeper feature of the language, are very apparent, for example, when you'd like to alter something inside of them:

```
IDL> h=hash('a',{b:1,c:2})
IDL> print,h['a'].b
      1
IDL> h['a'].b=2
% Attempt to store into an expression: Structure reference.
% Execution halted at: $MAIN$            84
```

The same happens for arrays stored in hashes. You're required to copy the entire array or structure out, modify it, then copy it back into the hash (remind anyone of the old days and widget state?). If it were possible for HASH dereferencing to return true IDL variable references, they would be far more useful.

Any workarounds to this "write-only" HASH behavior (other than, say, "use a pointer")?

JD

Subject: Re: Modifying Arrays and Structures in HASH's (hint: you can't)
Posted by Lajos Foldy on Sun, 28 Jul 2013 19:32:25 GMT
View Forum Message <> Reply to Message

On Sunday, July 28, 2013 6:13:18 AM UTC+2, bobnn...@gmail.com wrote:

> I will say that the inability of accessing directly into hashes and lists (by reference, not copy) is so disappointing that I am moving away from IDL. So if you found a way around this it would be very nice.

LIST is a pointer array in disguise. You can get a copy of this array and manipulate list elements directly:

```
IDL> l=list(1,2,3)
IDL> a=l.idl_container::get(/all)
IDL> print, l
      1
      2
      3
IDL> *a[1]=123
IDL> print, l
      1
```

123
3

regards,
Lajos

Subject: Re: Modifying Arrays and Structures in HASH's (hint: you can't)
Posted by m_schellens on Mon, 29 Jul 2013 08:12:46 GMT
View Forum Message <> Reply to Message

Am Sonntag, 28. Juli 2013 06:13:18 UTC+2 schrieb bobnn...@gmail.com:
> On Friday, July 26, 2013 11:24:01 AM UTC-6, mschellens wrote:
>
>> Am Freitag, 23. März 2012 00:24:08 UTC+1 schrieb pp.pe...@gmail.com:
>
>>
>
>>> It is not an intrinsic limitation of their objectyness. It is all about how the overloadbracketsleftside method is defined. Once someone pointed out that problem to me (in the case of arrays), I had several discussions with the developers (some in this newsgroup), and eventually the methods were changed (in 8.1, I think) so that through multiple indices it can be done with arrays:
>
>>
>
>>>
>
>>
>
>>>
>
>>
>
>>>
>
>>
>
>>> IDL> h=hash('a',[0,3,9])
>
>>
>
>>>
>
>>
>
>>> IDL> print,h['a']
>

>>
>
>>>
>
>>
>
>>>   0   3   9
>
>>
>
>>>
>
>>
>
>>> IDL> print,h['a',1]
>
>>
>
>>>
>
>>
>
>>>   3
>
>>
>
>>>
>
>>
>
>>> IDL> h['a',1]=-1
>
>>
>
>>>
>
>>
>
>>> IDL> print,h['a',1]
>
>>
>
>>>
>
>>
>
>>>   -1
>

>>
>
>>>
>
>>
>
>>>
>
>>
>
>>>
>
>>
>
>>> The problems shows up in structures and array indices if they are used in the way written above:
>
>>
>
>>>
>
>>
>
>>>
>
>>
>
>>>
>
>>
>
>>> IDL> print,(h['a'])[1]
>
>>
>
>>>
>
>>
>
>>>   -1
>
>>
>
>>>
>
>>
>
>>> IDL> (h['a'])[1]=99

>
>>
>
>>>
>
>>
>
>>> % Expression must be named variable in this context: <INT Array[3]>.
>
>>
>
>>>
>
>>
>
>>> % Execution halted at: $MAIN$
>
>>
>
>>>
>
>>
>
>>>
>
>>
>
>>>
>
>>
>
>>> And that is all fault of the parser's weird use of values instead of references on qualified names. Which, for that reason, and for breaking the least surprise principle, should be changed in a future version (a compile_opt, or a new file extension, to keep compatibility).
>
>>
>
>>>
>
>>
>
>>>
>
>>
>
>>>
>
>>

>
>>> But I think that the multiple indices for arrays approach could easily be applied to structures, just making a small change to the overloadbracketsleftside method. I may write a small derived class for that purpose. In that way, it would be possible to do
>
>>
>
>>>
>
>>
>
>>>
>
>>
>
>>>
>
>>
>
>>> h['a','b']=2
>
>>
>
>>>
>
>>
>
>>>
>
>>
>
>>>
>
>>
>
>>>
>
>>
>
>>>
>
>>
>
>>> On Thursday, March 22, 2012 6:58:19 PM UTC-3, JDS wrote:
>
>>
>
>>>

>
>>
>
>>>> HASH's are nice in that they can contain any variable type, and make iterating over deeply nested data structure more humane. On the other hand, their limitations as separate standalone objects, and not as a deeper feature of the language, are very apparent, for example, when you'd like to alter something inside of them:
>
>>
>
>>>
>
>>
>
>>>>
>
>>
>
>>>
>
>>
>
>>>> IDL> h=hash('a',{b:1,c:2})
>
>>
>
>>>
>
>>
>
>>>> IDL> print,h['a'].b
>
>>
>
>>>
>
>>
>
>>>>   1
>
>>
>
>>>
>
>>
>
>>>> IDL> h['a'].b=2
>

>>
>
>>>
>
>>
>
>>>> % Attempt to store into an expression: Structure reference.
>
>>
>
>>>
>
>>
>
>>>> % Execution halted at: $MAIN$ 84
>
>>
>
>>>
>
>>
>
>>>>
>
>>
>
>>>
>
>>
>
>>>> The same happens for arrays stored in hashes. You're required to copy the entire array or structure out, modify it, then copy it back into the hash (remind anyone of the old days and widget state?). If it were possible for HASH dereferencing to return true IDL variable references, they would be far more useful.
>
>>
>
>>>
>
>>
>
>>>>
>
>>
>
>>>
>
>>

>
>>>> Any workarounds to this "write-only" HASH behavior (other than, say, "use a pointer")?
>
>>
>
>>>
>
>>
>
>>>>
>
>>
>
>>>
>
>>
>
>>>> JD
>
>>
>
>>
>
>>
>
>>
>
>>
>
>> I revive this thread now, because it came to my interest only recently (guess why).
>
>>
>
>>
>
>>
>
>> I think I found a possibility to store in-place in IDL objects, which is fully compatible to to-date code. And it looks and feels like it should be:
>
>>
>
>>
>
>>
>
>> e. g. (not working now)
>

>>
>
>>
>
>>
>
>> IDL> h=hash('a',{b:intarr(5),c:2})
>
>>
>
>> IDL> h['a'].b[1:3] = [2,2,7]
>
>>
>
>>
>
>>
>
>> Just signal to the OBJECT::_OVERLOADBRACKETSLEFTSIDE the struct access by setting OBJREF to a variable with a value of !NULL (must be done automatically by the IDL environment. Note: OBJREF is usually set to an instance of the SELF object.)
>
>>
>
>> OBJECT::_OVERLOADBRACKETSLEFTSIDE then sets OBJREF to a PTR to the element accessed. The IDL environment must then left-struct-access the heap variable.
>
>>
>
>> This only works for
>
>>
>
>> a) heap variables,
>
>>
>
>> which is fine as all HASH and LIST elements are heap variables.
>
>>
>
>> b) Single elements (it could be extended easily by returning a PTR array).
>
>>
>
> Maybe you could post some code, as I cannot follow your description.
>
>

>
> I will say that the inability of accessing directly into hashes and lists (by reference, not copy) is so disappointing that I am moving away from IDL. So if you found a way around this it would be very nice.
>
>
>
>>
>
>>
>
>> This can be done even with any (IDL_OBJECT derived) object as no internal tricks are needed in _OVERLOADBRACKETSLEFTSIDE.
>
>>
>
>>
>
>>
>
>> (Of course this is also the way it is done in GDL :-)
>
>>

I should have been more expilcit:
Currently, you cannot use my suggestion, as it requires some changes within IDL.

I wanted to point out, that it is possible to extend IDL in a 'natural' (and backwards-compatible) way.
EXELIS should incorporate this in the next IDL version (or ASAP). Because currently this is a clear lack of IDL.

And in GDL the feasibility is demonstrated.

Marc

Subject: Re: Modifying Arrays and Structures in HASH's (hint: you can't)
Posted by m_schellens on Mon, 29 Jul 2013 08:46:36 GMT
View Forum Message <> Reply to Message

Am Sonntag, 28. Juli 2013 21:32:25 UTC+2 schrieb fawltyl...@gmail.com:
> On Sunday, July 28, 2013 6:13:18 AM UTC+2, bobnn...@gmail.com wrote:
>
>
>
>> I will say that the inability of accessing directly into hashes and lists (by reference, not copy) is so disappointing that I am moving away from IDL. So if you found a way around this it would be

very nice.
>
>
>
> LIST is a pointer array in disguise. You can get a copy of this array and manipulate list elements directly:
>
>
>
> IDL> l=list(1,2,3)
>
> IDL> a=l.idl_container::get(/all)
>
> IDL> print, l
>
> 1
>
> 2
>
> 3
>
> IDL> *a[1]=123
>
> IDL> print, l
>
> 1
>
> 123
>
> 3
>
>
>
> regards,
>
> Lajos

As of IDL 8.0, this is not correct. An IDL LIST is really a sinlge linked list made up of (PTR) heap variable nodes (IDL_CONTAINER_NODE). The IDL_CONTAINER::GET function creates then the array.

But your method works, as the (copied) pointers access the same heap variables. This is also the core of the mechanism I suggested for _OVERLOADBRACKETSLEFTSIDE.

Also note, that at least with HEAP the IDL_CONTAINER::GET functionality cannot work anymore (as you cannot pick the right element).

And it is of course as well not efficient, to convert the complete container to a pointer array in

order to left-access one element.
And the call to GET is almost as ugly as copying out one element, left-accessing it and copying it back.

Regards,
Marc

Subject: Re: Modifying Arrays and Structures in HASH's (hint: you can't)
Posted by Lajos Foldy on Mon, 29 Jul 2013 09:44:14 GMT
View Forum Message <> Reply to Message

On Monday, July 29, 2013 10:46:36 AM UTC+2, mschellens wrote:

> As of IDL 8.0, this is not correct. An IDL LIST is really a sinlge linked list made up of (PTR) heap variable nodes (IDL_CONTAINER_NODE). The IDL_CONTAINER::GET function creates then the array.
>
> But your method works, as the (copied) pointers access the same heap variables. This is also the core of the mechanism I suggested for _OVERLOADBRACKETSLEFTSIDE.
>
> Also note, that at least with HEAP the IDL_CONTAINER::GET functionality cannot work anymore (as you cannot pick the right element).
>
> And it is of course as well not efficient, to convert the complete container to a pointer array in order to left-access one element.
>
> And the call to GET is almost as ugly as copying out one element, left-accessing it and copying it back.
>

With huge list elements, copying out and back is very unefficient, creating a pointer array is much faster.

I do not understand the idea behind list. If it is a linked list, then accessing elements through subscripting is O(n) vs. O(1) in arrays. This makes lists practically unusable.

Try this test program to access the last element in a list:

```
pro list_test
l=list(1,2,3)

t=systime(1)
for j=1,10l^6 do x=l[-1]
print, " 3 elements: ", systime(1)-t

for j=4,10l^3 do l.add, j
```

```
t=systime(1)
for j=1,10l^6 do x=l[-1]
print, " 10^6 elements: ", systime(1)-t

end
```

IDL 8.2.3:

```
 3 elements:        7.6518829
 10^6 elements:        47.781787
```

GDL CVS:

```
 3 elements:        0.5020251
 10^6 elements:        44.200854
```

FL 0.79.25:

```
 3 elements:      0.044948101
 10^6 elements:      0.042613983
```

My pointer array based LIST implementation is about 10-150x faster for the small list, and more than 1000x faster for the large list.

regards,
Lajos

Subject: Re: Modifying Arrays and Structures in HASH's (hint: you can't)
Posted by m_schellens on Mon, 29 Jul 2013 11:24:01 GMT
View Forum Message <> Reply to Message

Am Montag, 29. Juli 2013 11:44:14 UTC+2 schrieb fawltyl...@gmail.com:
> On Monday, July 29, 2013 10:46:36 AM UTC+2, mschellens wrote:
>
>
>
>> As of IDL 8.0, this is not correct. An IDL LIST is really a sinlge linked list made up of (PTR) heap variable nodes (IDL_CONTAINER_NODE). The IDL_CONTAINER::GET function creates then the array.
>
>>
>
>> But your method works, as the (copied) pointers access the same heap variables. This is also the core of the mechanism I suggested for _OVERLOADBRACKETSLEFTSIDE.
>
>>

>
>> Also note, that at least with HEAP the IDL_CONTAINER::GET functionality cannot work anymore (as you cannot pick the right element).
>
>>
>
>> And it is of course as well not efficient, to convert the complete container to a pointer array in order to left-access one element.
>
>>
>
>> And the call to GET is almost as ugly as copying out one element, left-accessing it and copying it back.
>
>>
>
>
>
> With huge list elements, copying out and back is very unefficient, creating a pointer array is much faster.
>
>
>
> I do not understand the idea behind list. If it is a linked list, then accessing elements through subscripting is O(n) vs. O(1) in arrays. This makes lists practically unusable.
>
>
>
> Try this test program to access the last element in a list:
>
>
>
> pro list_test
>
> l=list(1,2,3)
>
>
>
> t=systime(1)
>
> for j=1,10l^6 do x=l[-1]
>
> print, " 3 elements: ", systime(1)-t
>
>
>
> for j=4,10l^3 do l.add, j
>

>
>
> t=systime(1)
>
> for j=1,10l^6 do x=l[-1]
>
> print, " 10^6 elements: ", systime(1)-t
>
>
>
> end
>
>
>
> IDL 8.2.3:
>
>
>
> 3 elements: 7.6518829
>
> 10^6 elements: 47.781787
>
>
>
> GDL CVS:
>
>
>
> 3 elements: 0.5020251
>
> 10^6 elements: 44.200854
>
>
>
> FL 0.79.25:
>
>
>
> 3 elements: 0.044948101
>
> 10^6 elements: 0.042613983
>
>
>
>
>
> My pointer array based LIST implementation is about 10-150x faster for the small list, and more than 1000x faster for the large list.

>
>
>
> regards,
>
> Lajos

I would like to emphasize, that I revided this thread for the suggestion about _OVERLOADBRACKETSLEFTSIDE. This is not limited to a particular container type.
What do you think about it?

The strength of a LIST is the deletion and insertion of elements.
Particular at the beginning or at the end (O(1)).
Not the traversal, what you measured.
I am sure, one can build an example, where a list implementation based on an array will loose against a real linked list. What if you fill the complete LIST from the left (like: list.ADD,element[i],0)?
For an array based LIST, even as you demonstrate that it is for some cases more efficient, one could say: Why not using a PTR array then directly?
Ok, you got some comfort functions. Maybe there is even room (or need) for an array based container with ADD, REMOVE, ...
But I think if the user uses a LIST he possibly really want one.

Regards,
Marc

Subject: Re: Modifying Arrays and Structures in HASH's (hint: you can't)
Posted by m_schellens on Mon, 29 Jul 2013 11:25:48 GMT
View Forum Message <> Reply to Message

Am Montag, 29. Juli 2013 13:24:01 UTC+2 schrieb mschellens:
> Am Montag, 29. Juli 2013 11:44:14 UTC+2 schrieb fawltyl...@gmail.com:
>
>> On Monday, July 29, 2013 10:46:36 AM UTC+2, mschellens wrote:
>
>>
>
>>
>
>>
>

>>> As of IDL 8.0, this is not correct. An IDL LIST is really a sinlge linked list made up of (PTR) heap variable nodes (IDL_CONTAINER_NODE). The IDL_CONTAINER::GET function creates then the array.
>
>>
>

>>>
>
>>
>
>>> But your method works, as the (copied) pointers access the same heap variables. This is also the core of the mechanism I suggested for _OVERLOADBRACKETSLEFTSIDE.
>
>>
>
>>>
>
>>
>
>>> Also note, that at least with HEAP the IDL_CONTAINER::GET functionality cannot work anymore (as you cannot pick the right element).
>
>>
>
>>>
>
>>
>
>>> And it is of course as well not efficient, to convert the complete container to a pointer array in order to left-access one element.
>
>>
>
>>>
>
>>
>
>>> And the call to GET is almost as ugly as copying out one element, left-accessing it and copying it back.
>
>>
>
>>>
>
>>
>
>>
>
>>
>
>> With huge list elements, copying out and back is very unefficient, creating a pointer array is much faster.
>
>>

>
>>
>
>>
>
>> I do not understand the idea behind list. If it is a linked list, then accessing elements through subscripting is O(n) vs. O(1) in arrays. This makes lists practically unusable.
>
>>
>
>>
>
>>
>
>> Try this test program to access the last element in a list:
>
>>
>
>>
>
>>
>
>> pro list_test
>
>>
>
>> l=list(1,2,3)
>
>>
>
>>
>
>>
>
>> t=systime(1)
>
>>
>
>> for j=1,10l^6 do x=l[-1]
>
>>
>
>> print, " 3 elements: ", systime(1)-t
>
>>
>
>>
>

```
>>
>
>>  for j=4,10l^3 do l.add, j
>
>>
>
>>
>
>>
>
>>  t=systime(1)
>
>>
>
>>  for j=1,10l^6 do x=l[-1]
>
>>
>
>>  print, " 10^6 elements: ", systime(1)-t
>
>>
>
>>
>
>>
>
>>  end
>
>>
>
>>
>
>>
>
>>  IDL 8.2.3:
>
>>
>
>>
>
>>
>
>>   3 elements:        7.6518829
>
>>
>
>>   10^6 elements:        47.781787
>
```

>>
>
>>
>
>>
>
>> GDL CVS:
>
>>
>
>>
>
>>
>
>> 3 elements: 0.5020251
>
>>
>
>> 10^6 elements: 44.200854
>
>>
>
>>
>
>>
>
>> FL 0.79.25:
>
>>
>
>>
>
>>
>
>> 3 elements: 0.044948101
>
>>
>
>> 10^6 elements: 0.042613983
>
>>
>
>>
>
>>
>
>>
>

>>
>
>> My pointer array based LIST implementation is about 10-150x faster for the small list, and more than 1000x faster for the large list.
>
>>
>
>>
>
>>
>
>> regards,
>
>>
>
>> Lajos
>
>
>
> I would like to emphasize, that I revided this thread for the suggestion about _OVERLOADBRACKETSLEFTSIDE. This is not limited to a particular container type.
>
> What do you think about it?
>
>
>
> The strength of a LIST is the deletion and insertion of elements.
>
> Particular at the beginning or at the end (O(1)).
>
> Not the traversal, what you measured.
>
> I am sure, one can build an example, where a list implementation based on an array will loose against a real linked list. What if you fill the complete LIST from the left (like: list.ADD,element[i],0)?
>
> For an array based LIST, even as you demonstrate that it is for some cases more efficient, one could say: Why not using a PTR array then directly?
>
> Ok, you got some comfort functions. Maybe there is even room (or need) for an array based container with ADD, REMOVE, ...
>
> But I think if the user uses a LIST he possibly really want one.
>
>
>
> Regards,
>

> Marc

Sorry, you did not measure the traversal, but the access to the last element (which is even worse for lists)

Subject: Re: Modifying Arrays and Structures in HASH's (hint: you can't)
Posted by Bob[4] on Mon, 29 Jul 2013 14:34:50 GMT
View Forum Message <> Reply to Message

On Sunday, July 28, 2013 1:32:25 PM UTC-6, fawltyl...@gmail.com wrote:
> On Sunday, July 28, 2013 6:13:18 AM UTC+2, bobnn...@gmail.com wrote:
>
>
>
>> I will say that the inability of accessing directly into hashes and lists (by reference, not copy) is so disappointing that I am moving away from IDL. So if you found a way around this it would be very nice.
>
>
>
> LIST is a pointer array in disguise. You can get a copy of this array and manipulate list elements directly:
>
>
>
> IDL> l=list(1,2,3)
>
> IDL> a=l.idl_container::get(/all)
>
> IDL> print, l
>
> 1
>
> 2
>
> 3
>
> IDL> *a[1]=123
>
> IDL> print, l
>
> 1
>
> 123
>
> 3
>

>
>
> regards,
>
> Lajos

That is nice, but it is really for hashes that I want this behavior. This trick does not work for them.

As an aside I saw a list of future IDL features and Dictionary type was on it. Lets hope that this is implemented correctly and not with the crippled operator overloading system in IDL (like hash is).

Bob

Subject: Re: Modifying Arrays and Structures in HASH's (hint: you can't)
Posted by Lajos Foldy on Mon, 29 Jul 2013 18:32:54 GMT
View Forum Message <> Reply to Message

On Monday, July 29, 2013 1:24:01 PM UTC+2, mschellens wrote:

> I would like to emphasize, that I revided this thread for the suggestion about _OVERLOADBRACKETSLEFTSIDE. This is not limited to a particular container type.
>
> What do you think about it?

```
FL> h=hash('s', {i:0})
FL> h['s'].i=123
FL> print, h
s: {     123}
FL>
FL> l=list({i:0})
FL> l[0].i=123
FL> print, l
{     123}
```

I added these features when HASH and LIST were implemented, so I agree with you, this should be the normal behavior for HASH, LIST and any future container.

> The strength of a LIST is the deletion and insertion of elements.
> Particular at the beginning or at the end (O(1)).

LIST::add, remove and access (subscripting) both have an index parameter, so the interface is that of a random access container, while the implementation is a sequential one. If the index parameter is there, users will use it :-)

> Not the traversal, what you measured.
>
> I am sure, one can build an example, where a list implementation based on an array will loose

against a real linked list. What if you fill the complete LIST from the left (like: list.ADD,element[i],0)?

Yes, this will be slow with an array implementation, but can be easily cured with a deque.

>
> For an array based LIST, even as you demonstrate that it is for some cases more efficient, one could say: Why not using a PTR array then directly?

The array management is hidden in LIST.

> Ok, you got some comfort functions. Maybe there is even room (or need) for an array based container with ADD, REMOVE, ...

> But I think if the user uses a LIST he possibly really want one.

Are you sure? :-)

regards,
Lajos

Subject: Re: Modifying Arrays and Structures in HASH's (hint: you can't)
Posted by David Fanning on Mon, 29 Jul 2013 18:37:01 GMT
View Forum Message <> Reply to Message

fawltylanguage@gmail.com writes:

>> But I think if the user uses a LIST he possibly really want one.
>
> Are you sure? :-)

Amen to this! ;-)

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.idlcoyote.com/
Sepore ma de ni thue. ("Perhaps thou speakest truth.")

Subject: Re: Modifying Arrays and Structures in HASH's (hint: you can't)
Posted by m_schellens on Mon, 29 Jul 2013 22:46:21 GMT
View Forum Message <> Reply to Message

Am Montag, 29. Juli 2013 20:32:54 UTC+2 schrieb fawltyl...@gmail.com:
> On Monday, July 29, 2013 1:24:01 PM UTC+2, mschellens wrote:
>
>
>
>> I would like to emphasize, that I revided this thread for the suggestion about _OVERLOADBRACKETSLEFTSIDE. This is not limited to a particular container type.
>
>>
>
>> What do you think about it?
>
>
>
> FL> h=hash('s', {i:0})
>
> FL> h['s'].i=123
>
> FL> print, h
>
> s: {   123}
>
> FL>
>
> FL> l=list({i:0})
>
> FL> l[0].i=123
>
> FL> print, l
>
> {   123}
>
>
>
> I added these features when HASH and LIST were implemented, so I agree with you, this should be the normal behavior for HASH, LIST and any future container.

Good. So what about _OVERLOADBRACKETSLEFTSIDE ?
Does FL use the interface I suggested as well?
Would be in everybody's interest if all implementations are compatible.
Unless you have (or anybody else has) a better idea.

>> The strength of a LIST is the deletion and insertion of elements.
>

>> Particular at the beginning or at the end (O(1)).
>
>
>
> LIST::add, remove and access (subscripting) both have an index parameter, so the interface is that of a random access container, while the implementation is a sequential one. If the index parameter is there, users will use it :-)
>
>> Not the traversal, what you measured.
>
>
>> I am sure, one can build an example, where a list implementation based on an array will loose against a real linked list. What if you fill the complete LIST from the left (like: list.ADD,element[i],0)?
>
>
>
> Yes, this will be slow with an array implementation, but can be easily cured with a deque.
>

Just that there is none...

>>
>
>> For an array based LIST, even as you demonstrate that it is for some cases more efficient, one could say: Why not using a PTR array then directly?
>
>
>
> The array management is hidden in LIST.

I just pointed out that in IDL a LIST is not a wrapper for a PTR array.
An IDL expert would probably write an IDL program using LISTs, which might then perform not optimal under unexpected container time guarantees. He would possibly even derive from LIST...

>> Ok, you got some comfort functions. Maybe there is even room (or need) for an array based container with ADD, REMOVE, ...
>
>
>
>> But I think if the user uses a LIST he possibly really want one.
>
>
>
> Are you sure? :-)

Perfect! Not anymore :-)

Regards,
Marc

Subject: Re: Modifying Arrays and Structures in HASH's (hint: you can't)
Posted by Lajos Foldy on Tue, 30 Jul 2013 08:55:37 GMT
View Forum Message <> Reply to Message

On Tuesday, July 30, 2013 12:46:21 AM UTC+2, mschellens wrote:

> Good. So what about _OVERLOADBRACKETSLEFTSIDE ?
> Does FL use the interface I suggested as well?

No. HASH and LIST do not use _OVERLOADBRACKETSLEFTSIDE at all for performance reasons, subscripting is implemented at a lower level. _OVERLOADBRACKETSLEFTSIDE methods exist only for compatibility, they can be called directly, but they are much slower then subscripting.

> Would be in everybody's interest if all implementations are compatible.
> Unless you have (or anybody else has) a better idea.

IDL syntax and semantics are designed by Exelis, I don't think we can change that. We can only implement IDL better :-)

regards,
Lajos

Subject: Re: Modifying Arrays and Structures in HASH's (hint: you can't)
Posted by m_schellens on Tue, 30 Jul 2013 11:46:56 GMT
View Forum Message <> Reply to Message

Am Dienstag, 30. Juli 2013 10:55:37 UTC+2 schrieb fawltyl...@gmail.com:
> On Tuesday, July 30, 2013 12:46:21 AM UTC+2, mschellens wrote:
>
>
>
>> Good. So what about _OVERLOADBRACKETSLEFTSIDE ?
>
>> Does FL use the interface I suggested as well?
>
>
>
> No. HASH and LIST do not use _OVERLOADBRACKETSLEFTSIDE at all for performance reasons, subscripting is implemented at a lower level. _OVERLOADBRACKETSLEFTSIDE

methods exist only for compatibility, they can be called directly, but they are much slower then subscripting.
>
>
>
>> Would be in everybody's interest if all implementations are compatible.
>
>> Unless you have (or anybody else has) a better idea.
>
>
>
> IDL syntax and semantics are designed by Exelis, I don't think we can change that. We can only implement IDL better :-)

Well, as it is currently not available in IDL at all, my suggestion still stands alone. For the sake of IDL I hope Exelis will eventually implement it as well.

Regards,
Marc
